INFORMATION AND CONTROL 7, 151–179 (1964)

Labeling Schemata and Syntactic Descriptions of Pictures*

R. NARASIMHAN

Digital Computer Laboratory, University of Illinois, Urbana, Illinois

Advocating the view that the so-called pattern recognition problem is much more appropriately studied as a problem of pattern analysis and description, a specific syntactic descriptive model is proposed for classes of pictures composed of linelike elements. The implementation of this model envisages a two-stage processing of digitized input pictures. A formalism is worked out in detail within which the labeling algorithms used in the first stage can be efficiently described and studied. A system structure for a computer to carry out this class of picture processing operations—called parallel processing—is outlined and it is shown that a natural extension of ALGOL can be made to serve as an adequate programming language for such a computer. The extension of this model and the processing details to deal effectively with noisy input pictures is considered. Some criteria for acceptable noise-cleaning techniques are discussed. Several examples of labeled pictures are included to illustrate the validity and power of the proposed model and the processing techniques.

I. INTRODUCTION

This paper is concerned with a descriptive scheme for processing digitized pictures[1] with immediate application to the recognition of patterns occurring in bubble chamber negatives. This processing technique is actually much wider in scope and can be applied with equal advantage to other patterns made up of linelike elements: machine and hand-printed alphanumeric characters, certain classes of biological slides, etc. The lines in the pictures are not required to be of unit thickness or even of uniform thickness; moreover, the processing is essentially independent of the size, position, and orientation of the input presenta-

* This work is supported in part by Contract No. AT(11-1)-1018 with the U. S. Atomic Energy Commission.

[1] See Section II, A below for more formal definitions.

tion of the pictures. Accordingly, the class of patterns for which the techniques we develop apply are in fact quite large.

Briefly, the descriptive scheme envisaged can be summarized as follows: We assume that our recognition procedure is applied to a class of well-defined patterns. Let each physical realization of a pattern be referred to as a picture. We confine our considerations to pictures consisting of "thin," linelike elements and shall refer to these lines as roads to emphasize that they are not required to be of unit thickness. The processing for describing a picture (and thus recognizing the pattern it represents) is now carried out in two phases. In the first phase the input picture is converted to a labeled graph—with labeled vertices and branches—by means of certain well-defined labeling algorithms. This labeling is intended to identify neighborhoods with specified topological properties such as terminals, junctions, crossings, bends, corners, etc. In the second phase the labeled graph so obtained is reduced to a network composed of certain primitive strings.[2] This reduction is sought to be accomplished by making use of given "grammar rules" for string manipulation. These grammar rules, defined in terms of the labels generated in the first phase, would naturally contain in themselves an implicit characterization of the *class* of patterns under consideration.

While, ultimately, the adequacy of any model such as this has to be based on extensive, empirical verifications, it might be of some interest to consider the following plausibility argument: The "pictorial" correspondence between a picture composed of linelike elements and a graph consisting of vertices and branches is intuitively evident. That, in fact, an abstract graph with labeled vertices and branches can be constructed in a uniform manner needs substantiation; the explicit processing methods we describe in this paper will verify this. The connection matrix of this abstract graph, which describes the graph, is clearly also a description of the underlying picture. By making use of the vertex labels, the original matrix can be replaced by a reduced version incorporating only a subset of the vertices. (For example, in the reduced version one might choose to retain only the terminals and those junctions where three or more roads meet and discard all the bends and corners.) The matrix entries will now be strings of labeled branches rather than single branches. The grammar rules can now be applied to

[2] Actually, these are intended to group into certain standard categories the various branch configurations that occur in the graphs. See below in the next paragraph for a more informal discussion.

transform and compose these strings into certain primitive categories (or types). Finally, the matrix can be rewritten to represent a network made up of these primitive strings. For a given *class of pictures*, the total description scheme, then, is built out of the following: (1) an alphabet of primitive strings, and (2) a set of "well-formed" networks of these primitive strings. Any given input picture is now described in terms of a statement that it consists of such and such primitive strings put together in such and such a network.

This rather close analogy to language models and to the hierarchic levels in linguistic analysis is more than incidental. For this reason in an earlier report (Narasimhan, 1962) we referred to this model as a linguistic approach to pattern recognition. To demonstrate the adequacy of this model, that report contained several examples of input pictures processed according to the details set forth in the scheme, making use of a few very rudimentary grammar rules. We also presented there a variety of arguments to show why, in our opinion, it is much more appropriate to view the so-called pattern recognition problem as really the problem of pattern analysis and description and emphasized that the aim of any adequate recognition procedure should be not merely to arrive at a "yes," "no," or "don't know" decision but to produce a structured description of the input picture. It is our contention that no model can hope to accomplish this in any satisfactory way unless it has built into it, in some sense, a generative grammar for the class of patterns it is set up to analyze and recognize.

The structure of a recognition program based on the above model (with appropriate modifications to take account of the fact that actual pictures are seldom well-formed) has been worked out for scanning bubble chamber negatives (Narasimhan and Mayoh, 1963). This program is currently being coded and will be reported on at some future date. It might be of some interest to note here, however, that the second phase of this program is very similar in structure to the familiar syntax compilers used with artificial languages.

In this paper we shall restrict our considerations to the labeling phase of the descriptive scheme. Our principal concern will be to develop a formalism within which labeling algorithms can be adequately described and studied. These algorithms will be defined in terms of computations of certain well-defined functions whose arguments and values are pictures. It will thus develop that picture processing of this type—which we shall refer to as parallel processing of pictures—is most naturally

carried out in a computer especially organized to handle a picture as a single structured operand in very much the same way as a conventional arithmetic computer is organized to treat a number as a structured operand. We shall outline very roughly the structure of such a computer and indicate how a suitably extended version of ALGOL could be made to serve as a programming language for this computer. These details together with examples of a few labeled pictures will be found in Section II.

In all the foregoing, for the most part, we have tacitly assumed that the input pictures are more or less well-formed. It is clear that in actual practice the normal situation is quite the reverse of what we have assumed. Thus, to have any practical significance, the scope of the theory has to be extended to include recognizably noisy pictures. A necessary feature of an adequate theory of pattern recognition is that it should be capable of suggesting techniques for noise cleaning which are optimal for the theory under consideration. In Section III of this paper we shall treat this problem in some detail and formulate some criteria for acceptable preprocessing routines. We shall see that these preprocessing algorithms fall strictly within the framework of the parallel processing formalism developed in the earlier part of the paper and hence can be carried out equally efficiently in a computer of the type referred to earlier.

The works of Selfridge, Dineen, Sherman, Unger and others in programming computers for picture processing are well-known by now. (For a comprehensive review of these see Minsky (1961).) Our indebtedness to all this work should be clear from the sequel. Concerning syntactic models for pattern recognition, however, we know of only three publications (Eden, 1961, 1962; Grimsdale *et al.*, 1959).[3] It is our view that the model based on labeling algorithms as described in this paper is not only different but that it offers intrinsically much greater potentialities for picture processing.

[3] Commenting on this statement, Professor Murray Eden has drawn our attention to the work on picture processing being carried at N. B. S. A general description of this project may be found in a report by R. A. Kirsch *et al.* in *Proc. EJCC* (1957). Dr. Eden remarks that more recent work by this group is reported in the following: D. Cohen, Picture processing in a picture language machine. Natl. Bur. Std. (U. S.) Rept. No. 7855 (April, 1962); R. Kirsh, Application of automata theory to problems in information processing. Natl. Bur. Std. (U. S.) Rept. No. 7882 (March, 1963).

II. A FORMALISM FOR LABELING SCHEMATA

A. The Formalism of Computation

The basic domain of operation for parallel processing is a fixed, finite array of points. Without loss of generality we shall assume this to be a square array of points. Let P be a generic symbol denoting the points of this array. P can assume values 0 or 1. A particular assignment of values to the points of the array constitutes a *picture*. Typically, in parallel processing, we start with an initial picture and generate from it, by well-defined operations, other pictures. In practice this is done by modifying the values of some of the P's in the old picture. It is important to note that, in general, we do not modify the values of all the points every time. Central to parallel processing, then, is the computation of functions defined over a square array of points, the values of the functions being dependent on specified pictures. In other words, the computation is done with respect to a specified picture. So, to specify a particular step in parallel processing, we have to specify a function, a picture and a set of points where the values of the function are to be computed with respect to this picture. However, within the parallel processing schema, we cannot operate on individual points or even on sets of points but only on the entire array of points. The only way we can refer to a particular set of points is by constructing a *characteristic picture* of that set, i.e., a picture in which the points of the set have the value 1 and all other points the value 0. Purely formally, then, we can say that each step in parallel processing consists in computing a given function with respect to two pictures—the *argument* and the *context* respectively—and generating a third picture.

To carry out such a computation we need at least three replicas of the basic square array. We will obviously need many more replicas to perform a sequence of such computations. Let us, then, visualize an entire column of such arrays, stacked one on top of another and assume that computations are performed within this setup. This already provides a preliminary specification for a system structure to realize these computations. But we shall postpone till the end of this section further consideration of the details of this problem.

Let S be a generic symbol representing these arrays and let particular pictures (i.e., specific arrays to whose points values have been assigned) be distinguished by the addition of subscripts.[4] A step in parallel

[4] It must be emphasized that this is just for notational convenience. These quantities should not be confused with subscripted variables in the sense

processing can now be formally represented by a statement of the form

$$S_k := F(S_i , S_j)$$

with the following convention: left hand side is the resulting picture; the first symbol in the function (on the right) refers to the argument set (i.e., its characteristic picture) and the second symbol to the context. Sometimes, in defining a picture, the context may be left unspecified. In all such cases it is understood that the context is the *universe*, i.e., the picture with all P values equal to 1. As we shall see presently, the computations will, in general, depend not only on the context picture but also on certain given parameters. Thus the general form of parallel processing computation should actually be represented as follows:

$$S_k := F(S_i , S_j ; \text{parameter}).$$

B. Some Special Functions for Picture Processing

Our next task is to define a set of functions in terms of which useful parallel processing routines can be developed. These functions will clearly form a hierarchy in the sense that some of them will be primitive, others defined in terms of these (i.e., as routines involving these), and still others in terms of this new set and so on. In designing a computer to realize these functions one would naturally want the machine orders to fit into this hierarchy at a level such as to provide a good match between the programming language and the machine language. We shall not, however, consider this problem in its complete generality here. In Appendix I we list a preliminary set of functions which we have arrived at on the basis of our work in picture processing. All the algorithms that have been developed so far can be efficiently described in terms of one or more of these functions. For our present discussion we restrict ourselves to a consideration of these functions and introduce some additional notations and definitions towards this end.

MARK AND CHAIN: For any point P, we shall denote by $N(P)$ the set of nine points consisting of P and its eight immediate neighbors in the array. The individual points of this set will be denoted by $N_i(P)$, $i = 0, 1, 2, 3, \cdots, 8$, where the subscript convention is as shown in

---

of ALGOL-60. We shall use the meta-linguistic notations of ALGOL wherever convenient.

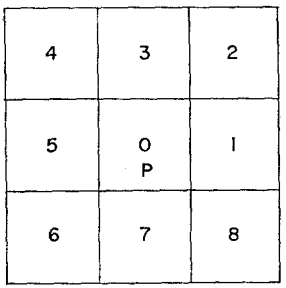

FIG. 1. Direction number convention

Fig. 1. By a direction list (or *direction* or *dir*, for short) we shall mean an index string $i_1 i_2 \cdots i_k$ with $k \leqq 9$ and each $i_j = 0, 1, \cdots$, or 8, no index value being repeated in the string. Using a direction list we can refer to a subset of $N(P)$ as follows:

$$\sum_{i \epsilon dir} N_i(P) \equiv \{Q \mid Q = N_i(P) \text{ for some } i \text{ in the direction list}\}.$$

If S is any picture, we can define a context-dependent subset of $N(P)$ as follows:

$$N(P, S) \equiv \{Q \mid Q \epsilon N(P) \textit{ and } Q \epsilon S\}.$$

Clearly we can extend this notion of context-dependence to subsets of $N(P)$ defined by a direction list.

In terms of these sets, the first two neighborhood operations listed in the appendix can be described.

$$\text{MARK } (S; \text{direction}) = \{Q \mid Q \epsilon \sum_{i \epsilon dir} N_i(P) \textit{ and } P \epsilon S\}.$$

$$\text{CMARK } (S_1, S_2; \text{direction})$$
$$= \{Q \mid Q \epsilon \text{ MARK } (S_1; \text{direction}) \textit{ and } Q \epsilon S_2\}.$$

Thus, the operation MARK marks all those immediate neighbors of the points in S specified in the direction list. CMARK is a context-dependent operation which selects a subset of the above marked set which is also included in the context picture.

Given a picture S and a point P in it, an *i-chain*, $(i = 1, 2, \cdots, 8)$, through P is the longest line of points $P_{k_1} P_{k_2} \cdots P_{k_m}$ such that $P_{k_j} \epsilon S$ for $j = 1, 2, \cdots, m$ and $P_{k_1} = P$ and $P_{k_j} = N_i(P_{k_{j-1}})$ for $2 \leqq j \leqq m$.

Clearly, an i-chain through P is a context dependent set. We shall denote it by $\omega_i(P, S)$. The operation CHAIN may now be described as follows:

CHAIN (S_1 , S_2 ; direction)

$$= \{Q \mid Q \in \omega_i(P, S_2) \text{ for some } i \in \text{dir } \textit{and} \text{ some } P \in S_1\}.$$

THRESHOLD: Denote by *Weight* $N(P, S)$, the number of points in $N(P, S)$ and by *length* $\omega_i(P, S)$, the length of the chain $\omega_i(P, S)$. We can now define the following new sets:

$$T(N(S_1 , S_2) \geqq m) \equiv \{P \mid P \in S_1 \textit{ and } \text{weight } N(P, S_2) \geqq m\};$$
$$T(\omega_i(S_1 , S_2) \geqq m) \equiv \{P \mid P \in S_1 \textit{ and } \text{length } \omega_i(P, S_2) \geqq m\}.$$

Here m is a positive integer. It is evident that these definitions can be extended to the general neighborhood operations MARK, CMARK, and CHAIN and to the other relational operators, $=$, $<$, $>$, $\leqq$. This extension yields us the general THRESHOLD operation as given in the appendix.

CONNECT: Given a picture S and a point P in it we define the kth iterate of $N(P, S)$, written as $[N(P, S)]^k$, in the following manner:

First, as an obvious extension of the notation $N(P, S)$, let us write $N(S_1 , S_2)$, where S_1 , S_2 are pictures, to denote the union of the sets $N(P, S_2)$ for all P in S_1 . The kth iterate is now given by the recursion,

$$[N(P, S)]^1 = N(P, S),$$
$$[N(P, S)]^k = N([N(P, S)]^{k-1}, S).$$

Define now a function

$$C(P, S) \equiv \{Q \mid \text{for some } k,\ Q \in [N(P, S)]^k\},$$

i.e., the set of all points Q such that for some k, Q is contained in the kth iterate of $N(P, S)$. Clearly, $C(P, S)$ defines the set of all points connected to P in the picture S. It is quite straightforward to extend this connectivity operation to the case where the function $N(P, S)$ is replaced by the more general neighborhood operation CMARK. It is easily verified that CHAIN is actually a particular example of this more general connectivity operation. The operation CONNECT SET given in the appendix is readily composed out of the function $C(P, S)$ defined above.

ORDERING: Superimpose an x-y coordinate system on the square array in the obvious manner. Let $x(P)$, $y(P)$ denote the x, y coordinates

of P. Using these we can now order the points of the set S as follows: for $P_1, P_2 \in S$ we shall write $P_1 < P_2$ ("is smaller than") provided,

$$x(P_1) < x(P_2)$$

or

$$x(P_1) = x(P_2) \quad \textit{and} \quad y(P_1) < y(P_2).$$

We can now refer to the *first* (i.e., smallest) point of S and *last* (i.e., largest) point of S.

Many of the concepts of pointset topology can be borrowed for our use. For example, in a quite straightforward manner, we can define the notion of a simply-connected region, a rectangle, a rectangular cover for a set S and so on. It is clearly possible to define parallel processing operations involving these. However, for our immediate purpose these additional functions are not required and hence we shall not discuss their implementation here.

C. A System Structure for Machine Realization

In Section II,A above we have already seen that the most natural machine organization to realize parallel processing computations consists of a stack of two-dimensional registers (referred to as *planes*) arranged one on top of another in the form of a column, each being a replica of our basic domain of operation, i.e., the square array. In terms of the special functions we have been discussing in Section II,B, it is evident that the control unit associated with this stack should be capable of performing certain primitive set-operations and thresholding operations.

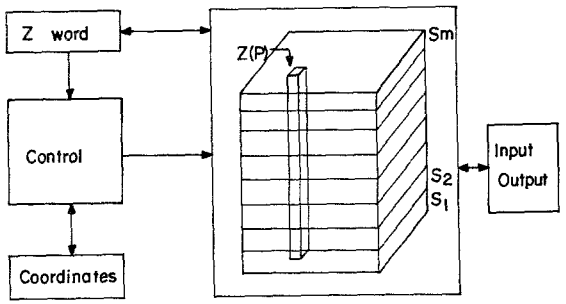

FIG. 2. Structure of a parallel processing computer

Consider now a stack consisting of m planes. Denote by $z(P)$, the m-bit word consisting of the point P in each plane (see Fig. 2). As has already been mentioned, in picture processing, typically, we start with the original picture in some plane (say, plane S_1) and, by well-defined algorithms, we assign labels to each point P of this picture (i.e., we generate a sequence of pictures S_{i_1}, S_{i_2}, $\cdots$, each corresponding to a particular computed label). Thus, at any moment during the processing, the word $z(P)$ contains all the presently generated labeling information about the point P. The natural thing to do would be to make use of this information in determining the course of the processing sequence. The facility required to be able to perform such conditional branching operations is the ability to read the word $z(P)$ for any specified point P. Closely related to this requirement is the ability to list the coordinates of a specified set S of points.

Figure 2 shows in a very rudimentary form a system organization for a parallel processing computer. This organization which we have arrived at from macroconsiderations is almost identical to a version originally proposed by Divilbiss and McCormick (1961) based on concepts and operations which enter in a natural way in bubble chamber recognition work. A considerably altered and very much improved version of this computer—called the Pattern Articulation Unit (PAU)—is currently under fabrication in this laboratory by a group led by B. H. McCormick. In this realization, for reasons concerned with hardware design and circuit simplification, the stack of planes is divided into two functionally distinct parts—a computing part (referred to as "stalactites"), and a storage part (referred to as the "transfer memory") from which the z-words are read out. (A description of the structure of the PAU as well as that of Illiac III, a general purpose pattern recognition computer, of which the PAU forms a part, may be found in McCormick (1963).)

D. A Programming Language for Parallel Processing

To complete the task of formalization of picture processing algorithms it is now necessary to describe a formal programming language in which a computer such as the one described above can be programmed. A natural approach would be to examine whether this formal language could be obtained as a suitable extension of some already well-developed programming language. We shall now show that this is readily accomplished by using ALGOL as a base language.

Table I gives an analysis of the structure of ALGOL-60 in terms of its syntactic constituents. In Table II we have indicated a possible extension of this language to incorporate computations involving pictures. The notation in this table corresponds to that in Table I and only the additions have been shown. As might have been expected, we have adjoined to the language *Picture* as a new structured operand and have included the primitive operations required to realize the functions we have so far considered. Thus a simple variable can identify a picture and must be declared to that effect in type declaration. It is clear that a picture expression can be defined completely analogous to an arithmetic expression. It is also clear that a Boolean expression can be simi-

TABLE I

ALGOL-60 STRUCTURE

1. Primitive symbols:	1. Letters	
	2. Digits	
	3. T-values	
	4. Punctuations	
2. Primitive operands:	1. Identifier	Simple variable, array, label, switch, procedure
	2. Number	
	3. String	
3. Primitive operations:	1. Arithmetic	$+ \; - \; \times \; / \; \div \; \uparrow$
	2. Relational	$< \; \leqq \; = \; > \; \geqq \; \neq$
	3. Logical	$\equiv \; \supset \; \vee \; \wedge \; \neg$
4. Primitive functions:	1. Arithmetic expression	Computes a number
	2. Boolean expression	Computes a T-value
	3. Designational Expression	Computes a label
5. Primitive subroutines:	1. Assignment statement	
	2. *Goto* statement	
	3. Conditional statement	
	4. *For* statement	
	5. Procedure statement	
	6. Dummy statement	
6. Routines:	1. Block	*Input specification* Declaration: 1. Type declaration 2. Array declaration 3. Switch declaration 4. Procedure declaration

TABLE II

EXTENSIONS TO PICTURE PROCESSING

2. Primitive operand:	1. Identifier	Picture, direction
3. Primitive operations:	4. Picture operations	Sum, product, complement mark, threshold, list, etc.
4. Primitive functions:	4. Picture expression	Computes a picture
		Boolean expression can be extended to include picture expressions; e.g., If clause: : = *If* (< picture expression> = 0) *then*
6. Routine:		*Input specifications:*
		1. Type declaration: := real | integer | Boolean | picture
		5. Direction declaration

larly extended to incorporate picture expressions. We shall not consider the formal details here.

Direction is introduced in the language somewhat like a switch. The following formal definitions should make this clear:

⟨direction identifier⟩ : : = ⟨identifier⟩
⟨direction number⟩ : : = 0|1|2|3|4|5|6|7|8
⟨direction list⟩ : : = ⟨direction number⟩ |
⟨direction list⟩ ⟨direction number⟩
⟨direction declaration⟩ : : = *direction* ⟨direction identifier⟩

Values can be assigned to a direction by means of an assignment statement in the usual way:

direction identifier : = direction list

It will be noticed that we have made no special provision to avoid repetitions in the direction list. This, however, is of no great consequence since the semantics can be set up to ignore any such repetitions that might occur. Also, to describe the "components" of a direction individually, it might be of assistance to introduce an additional definition:

⟨direction component⟩ : : = ⟨direction identifier⟩ [⟨direction number⟩].

Clearly, a required semantic restriction is that a direction component exists for a particular direction identifier only if the corresponding direc-

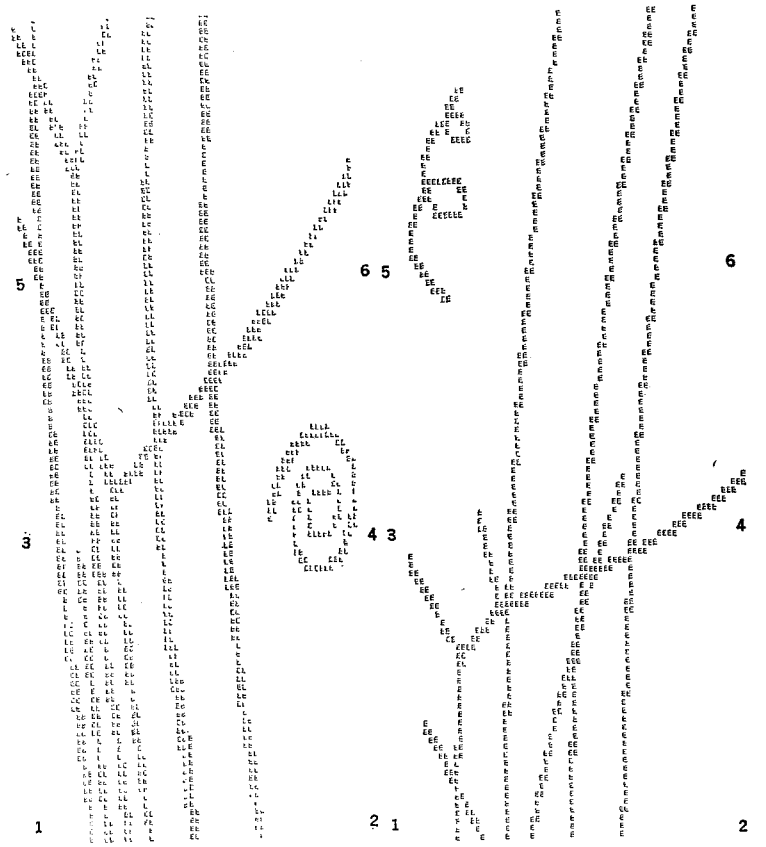

FIG. 3. Parts of two bubble chamber negatives: digitized, input pictures

tion number is included in the direction list that has been assigned to that identifier.

It is readily verified that this extended ALGOL-60 is quite adequate for the formal specification of all the picture processing functions we have considered so far and the routines based on them.[5]

E. Some Examples of Labeled Pictures

As was mentioned in the Introduction, a complete recognition program for use with bubble chamber negatives is currently being worked

[5] See Appendix II for examples of some routines written in this language.

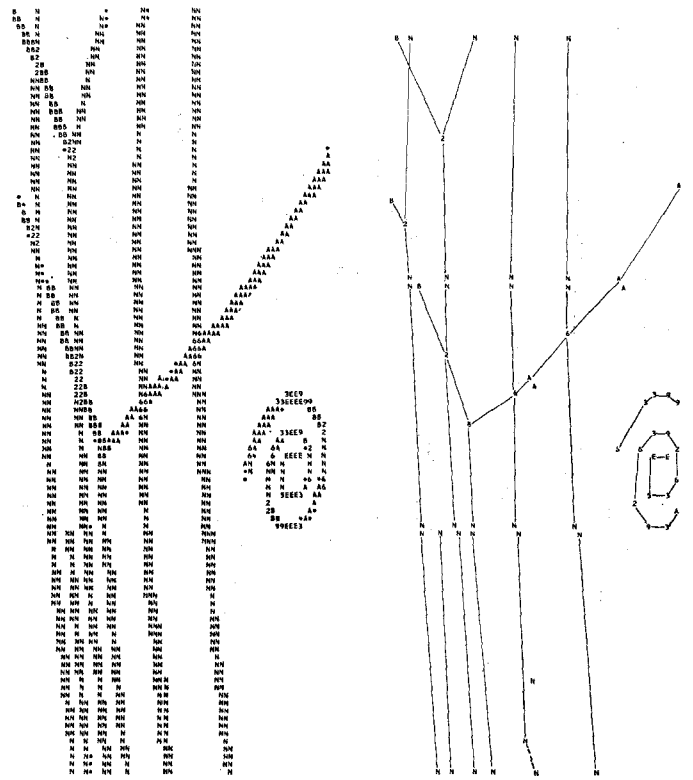

FIG. 4. Labeled output and abstracted graph of the picture on the left in Fig. 3.

out in this laboratory. As illustrative examples of the parallel processing schemata we have been considering so far, we shall give here a few outputs from the labeling algorithm associated with this program. These outputs were obtained using a parallel processing simulator that has been written for the IBM-7090 computer (see Appendix I).

Figure 3 shows the digitized versions of parts of two bubble chamber negatives. These were manually prepared from tracings of enlarged prints from the negatives. The pictures as shown are made up of an array of 100 × 68 bits each. Since the size of the basic plane in the simulator is only 32 × 32 bits, the input pictures were initially divided into six "windows" (32 × 32 bits each) as indicated in Fig. 3.[6] Each of

[6] Each window as actually used had an overlap of four bits with its right and

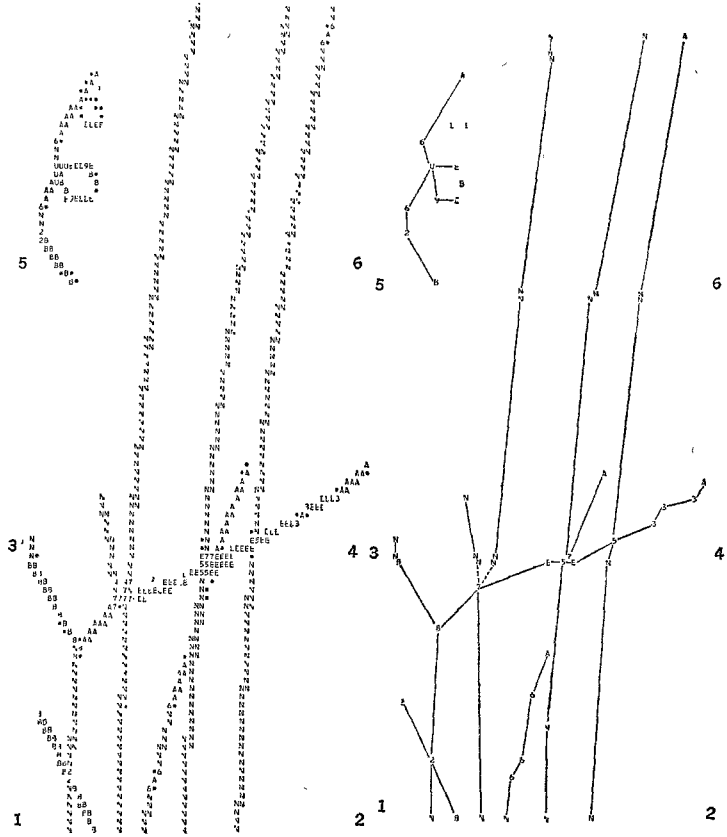

FIG. 5. Labeled output and abstracted graph of the picture on the right in Fig. 3.

these windows were processed independently and the outputs reassembled.

Figure 4 shows the labeled output and the "abstracted" graph for the first picture and Fig. 5, the two corresponding outputs for the second picture in Fig. 3. The four principal labels assigned to the branches are N (representing a north-south road); E (representing an east-west road); A (representing a right-diagonal road); and B (representing a left-diagonal road). The junctions, crossings, bends, etc., where two or

top neighbors and hence had a working field of 36 × 36 bits. In the "abstracted" graphs of Fig. 4 and Fig. 5, however, only the actual 32 × 32 windows are shown.

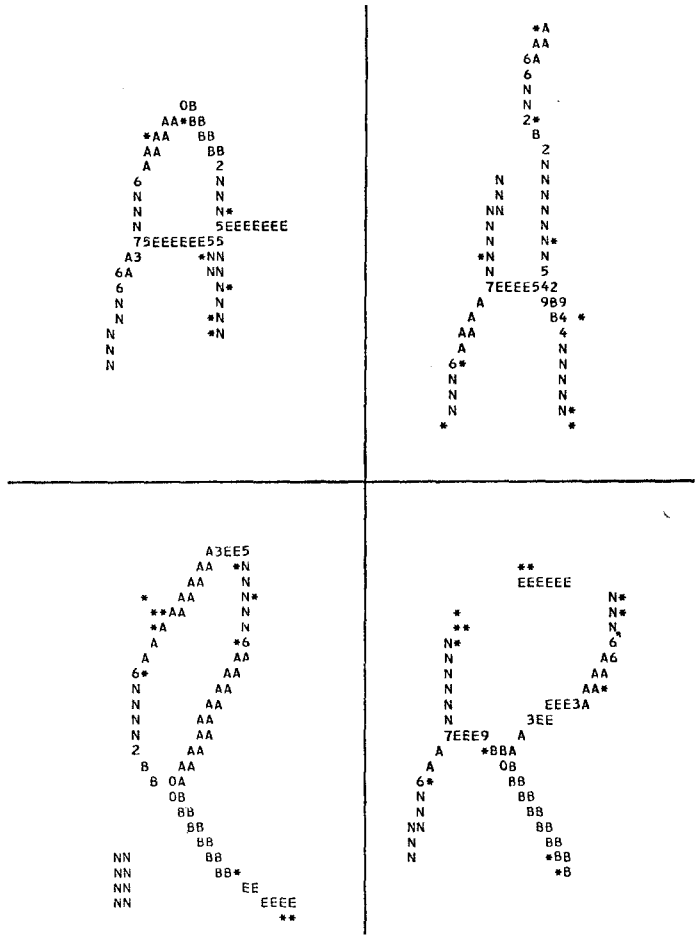

FIG. 6. Labeled output: sloppy, handprinted letters

more roads meet, are identified by their multiple labels. In the outputs these are given by the following code:

EA : 3	AB : 0	ENB : 4
EN : 5	NB : 2	ANB : 8
EB : 9	EAN : 7	EANB : U
AB : 6	EAB : 1	

The points which have not been assigned any of the labels are indicated by asterisks (*).

The graphs on the right-half of Fig. 4 and Fig. 5 were obtained by outputting a single representative point for each one of the multiple labeled sets and for the terminals. As mentioned earlier, this entire program was run independently once for each of the six windows in each input picture.

The full details of this labeling algorithm, since it forms part of a larger program, are not given here. However, in Appendix II we give that part of the labeling routine concerned with the initial assignment of the four primary labels. It will be seen from this that the picture processing functions discussed earlier and the programming language based on ALGOL do serve as an efficient device for describing our processing techniques.

These specific labeling algorithms were developed primarily for processing bubble chamber pictures. To determine to what extent they apply, without any modification, to other classes of patterns made up of linelike elements, the same program was run on a variety of other input pictures. Figure 6 shows the result of applying the labeling routine to four, "sloppy," handprinted letters.[7] It is evident that the algorithms remain valid for these pictures also.

Most of the published studies on recognition schemes for alphanumeric characters make use of "characteristic vectors" or "property lists" in an essential way. It is of interest to note that parallel processing of pictures based on labeling schemata offers a very efficient and powerful means of generating the input information required by these recognition procedures. This is clearly verified by the labeled outputs shown in Fig. 6.

III. NOISY PICTURES AND PREPROCESSING TECHNIQUES

A. Need for Preprocessing

As we pointed out in Section I, input pictures in any realistic situation are almost always noisy and a recognition procedure, to have practical significance, must take explicit cognizance of this fact and suggest constructive means of combating noise. However, to do this effectively it is necessary first to identify and delimit the types of noise which have to be contended against and then suggest optimal techniques for re-

[7] The input pictures were preprocessed before labeling, using gap filling and thinning routines. For details see Section III, C below.

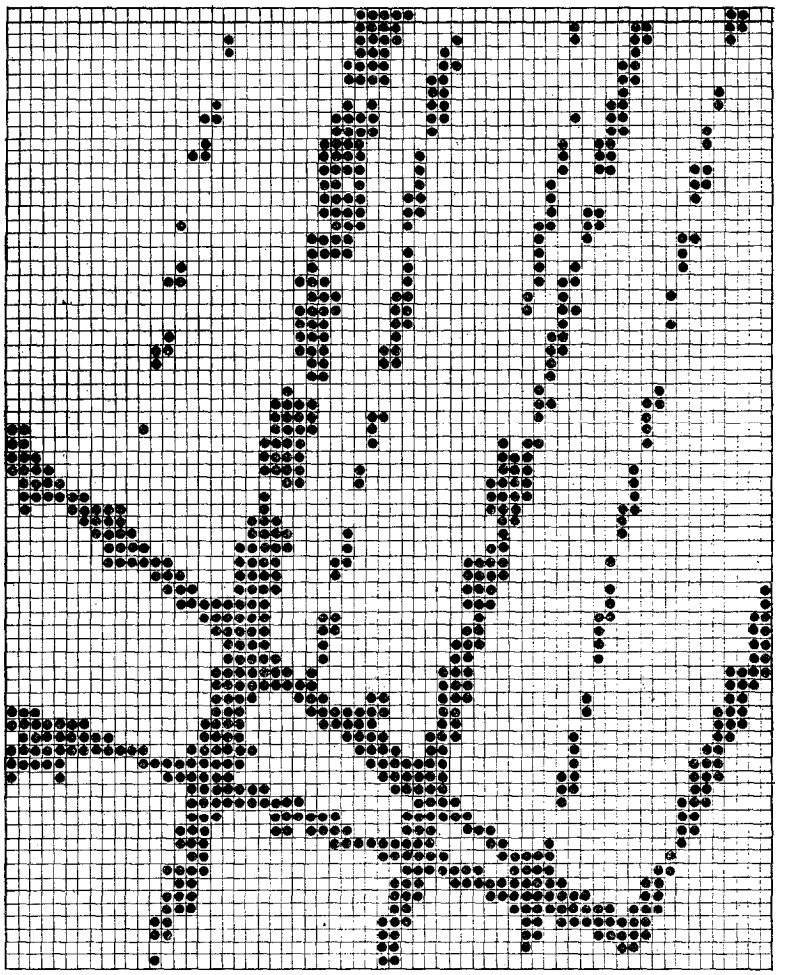

FIG. 7. Example of an unprocessed bubble chamber picture

moving the noise or at least for minimizing its effect. Clearly, both the delimitation and optimality, to be meaningful, must be specified with reference to a well-defined context. A necessary feature of an adequate theory is the ability to provide precisely such a context. In this section we shall study this problem from the point of view of syntactic models of description and suggest a few approaches to the solution of the noise-cleaning problem.

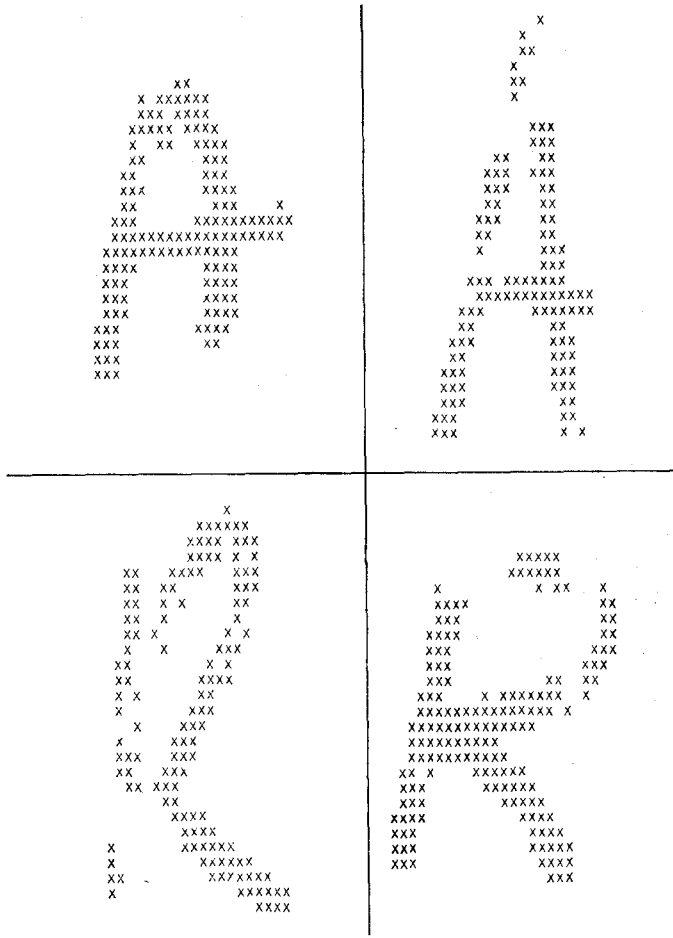

FIG. 8. Sloppy, handprinted letters: unprocessed

Figure 7 illustrates a fairly typical unprocessed bubble chamber picture. A few typical examples of "sloppy" hand-printed letters are shown in Fig. 8.[8] When these pictures are considered as representatives from the class of patterns that the model is designed to identify, it is evident that these contain two major types of distortions due to noise which one would like either to eliminate or at least minimize. These are (1)

[8] These samples are taken from Doyle (1960).

the occurrence of gaps in the picture and (2) the nonwellformedness of the roads that make up the picture. Although both these types of noise are not completely independent of each other, it is convenient to describe and study their effects separately.

The gaps that occur in a picture can be further divided into roughly three categories as follows: (a) isolated holes inside a road; (b) gaps along the edges of a road resulting in their uneven width; and (c) gaps which actually destroy basic connectivity in a picture. Several local averaging techniques are known to remove isolated gaps in a picture that occur in the midst of filled-in areas. (Cf., for instance, the paper by Dineen referred to earlier.) The particular labeling procedures considered in the last section do not require that the roads be of uniform width. Thus, for the most part, gaps of the first two types may be said to be syntactically not too harmful. However, the noise introduced by type (c) gaps can render the picture unidentifiable or ambiguous. "Large" gaps of this type can effectively be removed only by the use of the implicit syntax of the patterns. One approach could be based on the following technique: after a preliminary picture-independent gap filling, a cycle of "crude" labeling can be gone through. Next, using this first level labeling, a more syntax oriented gap filling can be attempted (see, for example, some of the labeled letters in Fig. 6). A second approach may be based on known physical properties of the process which generates the pictures. For example, in bubble chamber pictures most often gaps occur in beam tracks cutting them up into isolated line segments. Since the majority of beam tracks, however, have a north-south orientation, a method of bridging these gaps could be to extend the line segments preferentially in the north-south direction till the adjacent fragments meet. Labeling schemata can be used in a very powerful way to implement similar picture-dependent preprocessing techniques.

Nonwellformedness of roads can be due to the presence of either "fringes" or gaps at the edges of roads and is most often noticeable in the form of uneven and/or excessive thickness of the roads in relation to their length. Thinning is an attempt to minimize the distortion due to both these effects. Thinning procedures are of concern to our model since the labeling algorithms have been set up to identify roads on the basis of their length to width ratio; (see Appendix II for details). But it is worth pointing out here that, from our point of view, it is not suffi-

cient that thinning routines are set up to preserve local connectivity; rather, they should do so in a syntactically significant way.

B. Some Criteria for Acceptable Preprocessing Techniques

Within the framework of the parallel processing formalism outlined earlier, these preprocessing algorithms are strictly computations on pictures yielding other pictures. We shall refer to them generically as "transformations." Transformations based on specific Boolean operations performed in parallel on a given set of neighbors of each digitized point of a picture have been termed "homogenous logical transformations" elsewhere (Divilbiss and McCormick, 1961, p. 18). In terms of the function TRANSFORM, included in the list in Appendix I, such transformations are readily realized so long as the Boolean operations are performed only on the immediate neighbors. If it is necessary to use a larger neighborhood, the notation for the "direction list" has to be suitably extended. There are several obvious ways in which this can be done; however, we shall not go into the details here.

In practice most of these transformations would be applied as an iteration sequence. Hence they would have to satisfy suitable stability criteria—both asymptotic stability and stability with respect to the initial picture. Stability is a difficult notion to define quantitatively but its significance is easy enough to understand informally. A gap-filling transformation, when iterated sufficiently many times, should bring the picture to a state from which it does not change any further and, what is more, this final stable state should be one where the "essential" characteristics of the original picture are preserved. Analogous remarks hold for a thinning routine. A gap-filling routine that ultimately makes every point in the picture black or a thinning routine that ultimately makes every point in the picture white has very little practical value. This is so because, in general, when applying these transformations, the only feasible way to terminate their iteration is to base it on the binary decision whether on any given instance of the application of a transformation the state of the picture did or did not change.

Stability with respect to initial conditions would require that a transformation applied on two pictures, not too different to begin with, leaves them not too different at the end of the operation. Roughly speaking, this means that the transformation does not accentuate accidental

variations in the two pictures and allow the difference to grow with repeated application of the transformation. Clearly, this requirement is imperative if transformations are to remain insensitive to minor variations in the details of a picture; for example, those that arise as a result of differences in the digitizing procedure.

Where preprocessing consists in iterating a set of transformations rather than a single one, an additional requirement would be that the end result be independent of the specific order in which the transformations of the set are applied in any iteration step. This, perhaps, is a very strong condition to impose and it might be worthwhile to substitute some suitable, weaker restrictions.

For thinning routines, as we have already seen, the essential requirement is that the transformation should preserve the underlying syntax of the picture. Syntax-preserving thinning is a well-defined operation and easy to accomplish in the case of a picture with well-formed roads; in this case, it is sufficient to remove everything except a central strip along each road, an operation easily performed with local thresholding. (This has to be suitably modified, of course, to preserve basic connectivity where several roads meet or cross.) However, the problem becomes very complicated, as is readily verified, when the picture consists of nonwellformed roads.

C. Some Examples of Preprocessed Pictures

Several general purpose (i.e., picture independent) gap-filling and thinning routines have been developed in our work in picture processing. Figure 9 shows the result of the application of one such composite routine to a set of four letters. The input pictures used were the "sloppy," handprinted letters referred to earlier and shown in Fig. 8. The details of the preprocessing algorithm are given in Appendix II. A comparison between the unprocessed and processed pictures will verify that these particular noise-cleaning procedures are not only extremely efficient but preserve, in addition, the underlying syntax of the handwritten letters. The processed output letters shown in Fig. 9 were the ones used as input pictures for the labeling in Fig. 6.

IV. CONCLUDING REMARKS

In this paper we have been concerned with a specific model for describing pictures and with a particular class of processing procedures which we have termed "parallel processing." Labeling schemata based

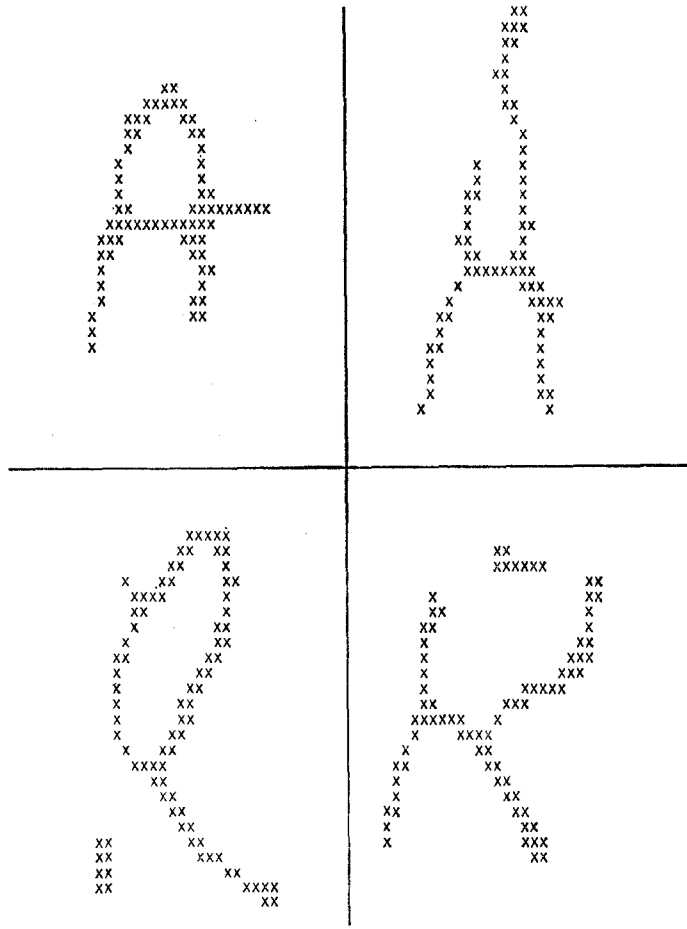

FIG. 9. Sloppy, handprinted letters: gap-filled and thinned

on parallel processing techniques have been shown, in our discussions and examples above, to offer a very natural and effective means of identifying the primary syntactic categories (i.e., the "alphabets") in terms of which pictures composed of linelike elements can be described and hence recognized. As mentioned earlier in the Introduction, a complete description scheme for a class of pictures would have to include, in addition to the alphabets, certain grammar rules to generate "well-formed" networks using these alphabets. Since the grammar

rules are intended to characterize implicitly the underlying patterns in the class of pictures, it is clear that those aspects of picture processing which are based on the grammar rules will, for maximum efficiency, have to depend on the input class of pictures. For this reason, such details are beyond the scope of this paper and have not been discussed here.

Although the particular labeling algorithms we have exemplified herein make use of the linelike aspects of the input picture in an essential way, the parallel processing formalism itself is of much wider scope and is, clearly, not limited in any intrinsic manner to pictures composed of roads. It is fairly easy to extend our labeling algorithms to apply to a wider class of pictures by incorporating in them suitable "bordering" (i.e., "contouring" or "profiling") operations. It is well-known that such operations can be realized in terms of quite straightforward threshold functions which are strictly within the scope of the parallel processing formalism. Studies along these lines are under way and we shall report on the results at a later date.

APPENDIX I

A LIST OF FUNCTIONS FOR PICTURE PROCESSING

(*Note:* S_1 , S_2 , S_3 are pictures, not necessarily different. For explanations about the functions, see text. A self-contained simulator for IBM-7090 incorporating all these functions and other red-tape operations has been written by J. H. Stein (1963).)

Group A: Set Operations

1. $S_1 := 0$
2. $S_1 := S_2$
3. $S_1 := \tilde{S}_2$
4. $S_1 := S_2 + S_3$
5. $S_1 := S_2 * S_3$
6. $S_1 := S_2 - S_3$ (i.e., $S_2 * \tilde{S}_3$)
7. $S_1 := S_2 \not\equiv S_3$ (i.e., $S_2 \oplus S_3$)

Group B: Neighborhood Operations

8. $S_1 :=$ MARK (S_2 ; direction)
9. $S_1 :=$ CMARK (S_2 , S_3 ; direction)
10. $S_1 :=$ CHAIN (S_2 , S_3 ; direction)
11. $S_1 :=$ TRANSFORM (S_2 , S_3 ; BFUNCTION)

(*Note:* 1. Direction is to be specified by a direction list of the form $i_1 i_2 \cdots i_k$ where $k \leqq 9$ and each $i_j = 0, 1, \cdots, 7$ or 8, without repetitions.

2. In (9) and (10), S_2 is the argument and S_3, the context.

3. In (11), BFUNCTION is any Boolean function with the direction numbers as the variables. Without loss of generality it may be assumed that the function is specified in the sum-of-products form. S_1 is a subset of S_2 and contains all those points of S_2 for which the BFUNCTION computed with the values as in S_3 is true.)

Group C: Threshold Operations

12. S_1 : = THRESHOLD (weight function; relational operator; m), where

    weight function :: = MARK | CMARK | CHAIN

    relational operator :: = $< \mid \leqq \mid = \mid \geqq \mid >$

    m is a positive integer.

(*Note:* In a threshold operation, *always*, $S_1 \subseteqq S_2$ where S_2 is the argument in the weight function.)

Group D: Boolean Operation

13. *If* $S_1 = 0$ *then*
14. *If* $S_1 \neq 0$ *then*

Group E: Connectivity Operation

15. CONNECT (P, S_2)
16. CONNECTSET (P, S_2 ; S_3)

(*Note:* P is a *source* point; S_2 is a tag or label. S_3 is a *sink*, usually a set of (connected) sets. In (14), all points in S_2 connected to P are marked. In (15), the members of S_3 connected to P by the label S_2 are listed. (See below for *list.*))

Group F: List Operation

17. LIST POINTS (S_1)
18. LISTSETS (S_1)

(*Note:* In (16), the coordinates of the points in S_1 are listed. In (17), S_1 is a set of (connected) sets and the operation lists one representative point for each set in S_1.)

Group G: Point Operation

19. Read $z(P)$
20. Write (P, S)

(*Note:* The point P is specified by giving its x-y coordinates. In (20), S is the name of some specific picture. The result of the WRITE operation is to set $P = 1$ in S.)

Group H: I/O Operation

21. Input
22. Output

(*Note:* These are picture input/output operations.)

APPENDIX II

In this appendix we give the details of part of the labeling algorithm and the gap-filling and thinning routines used in obtaining the output pictures included in the text. The algorithms are described in the form of procedures using the programming language discussed in Section II,D. We make explicit use of the functions listed in Appendix I. Picture operators are not indicated by any special symbol in as much as their use should be clear from the context.

LABELING

The following procedure describes the algorithm for assigning the primary labels N, E, A and B (see Section II,E for further explanations) to the input picture. The complete labeling routine contains in addition (i) a procedure to extend this first level assignment of labels; (ii) a procedure for vertex cleaning; (iii) a procedure for assigning direction numbers to vertices. These details will be published elsewhere at a later date.

```
Picture Procedure  LABEL (SI, SO, a, b, W, L);
        picture SI, SO; direction a, b;
        integer array W, L [1:3];
  begin picture S1, S2, S3;
        integer K;
        S1:= 0;
  for   K:= 1 step 1 until 3 do
  begin
```

$S2$:= THRESHOLD (CHAIN (SI, SI; b) $\leqq W[K]$);
$S3$:= THRESHOLD (CHAIN ($S2$, $S2$; a) $\geqq L[K]$);
$S1$:= $S1$ + $S3$ *end*;
SO := CHAIN ($S1$, $S1$; a) *end*

The following four procedure statements now assign the four labels N, E, A, B respectively using the actual parameters indicated. ORIG is the original input picture. NORTH, EAST, ALPHA, BETA are the respective labeled output pictures. The actual integer arrays used were:

$NW = EW = AW = BW = (1, 2, 3)$
$NL = (2, 4, 5)$
$EL = (2, 4, 6)$
$AL = BL = (3, 5, 6)$.

N: LABEL (ORIG, NORTH, 37, 15, NW, NL);
E: LABEL (ORIG, EAST, 15, 37, EW, EL);
A: LABEL (ORIG, ALPHA, 26, 48, AW, AL);
B: LABEL (ORIG, BETA, 48, 26, BW, BL).

PREPROCESSING

The actual gap-filling and thinning routines used to obtain the output pictures shown in Fig. 9 from the respective input pictures shown in Fig. 8 are given below, again, in the form of procedures. Both these preprocessing algorithms are stable (in the sense discussed in Section III) and hence can be iterated. In the program, first the gap-filling routine was used iteratively till stability was reached and then the thinning routine, again iteratively, up to a maximum of four times.

1. Gap-Filling

The following two procedures called CROSS and CORNER are set up to fill two types of gaps. In the actual program, first CROSS was performed and then CORNER. This sequence was iterated till stability was reached. In the description below, SI denotes the input picture and SO the resulting output picture. TRANS is used as an abbreviation for the function TRANSFORM (see Appendix I).

Picture Procedure CROSS (SI, SO);
picture SI, SO;

```
begin  SO: = ¬SI;
       SO: = TRANS (SO, SI; 15 + 26 + 37 + 48);
       SO: = SO + SI end

Picture Procedure  CORNER (SI, SO);
                   picture SI, SO;
  begin  picture S1, S2, S3, S4;
                   S1 := TRANS (SI, SI; 2 5̄ 6̄ 7̄);
                   S1 := MARK (S1; 123);
                   S2 := TRANS (SI, SI; 4 7̄ 8̄ 1̄);
                   S2 := MARK (S2; 345);
                   S3 := TRANS (SI, SI; 6 1̄ 2̄ 3̄);
                   S3 := MARK (S3; 567);
                   S4 := TRANS (SI, SI; 8 3̄ 4̄ 5̄);
                   S4 := MARK (S4; 781);
                   SO := SI + S1 + S2 + S3 + S4 end
```

2. Thinning

In the actual program, the thinning procedure described below was preceded by the gap filling routine CROSS in each iteration of the cycle. This was done to "smooth" out the resulting picture after each round of thinning. *SI* and *SO* again refer to the input and output pictures respectively.

```
Picture Procedure  THIN (SI, SO);
         picture SI, SO;
  begin  picture S1, S2, S3, S4, S5;
         S5 := THRESHOLD (CMARK (SI, SI; 12345678)
                ≦ 3);
         S1 := TRANS (SI, SI; 37);
         S2 := TRANS (SI, SI; 15);
         S1 := TRANS (S1, S1; 1 + 5);
         S2 := TRANS (S2, S2; 3 + 7);
         S3 := TRANS (SI, SI; 3 7̄ + 3̄ 7);
         S4 := TRANS (SI, SI; 1 5̄ + 1̄ 5);
         S3 := TRANS (S3, S1; 3 + 7);
         S4 := TRANS (S4, S2; 1 + 5);
         S1 := S3 + S4;
         SO := SI * S̄1̄;
```

$S2$:= THRESHOLD (CMARK ($S5$, $S1$; 12345678) = 2);
$S1$:= $S5 * \overline{S2}$;
SO := $SO + S1$ *end*

ACKNOWLEDGMENTS

Brian H. Mayoh and R. Kevin Rice have made essential contributions to the development of the labeling algorithms used here. In the machine realization of some of these algorithms, they were assisted by Larry Nelson and James Farnango. The parallel processing simulator for IBM-7090 was designed and coded by James H. Stein. It is a pleasure to acknowledge our indebtedness to all their help.

RECEIVED: June 5, 1963

REFERENCES

DIVILBISS, J. L. AND MCCORMICK, B. H. (1961), Tentative logical realization of a pattern recognition computer. Digital Computer Laboratory File No. 403, Univ. of Illinois.

DOYLE, W. (1960), Recognition of sloppy, handprinted characters. *Proc. Western Joint Computer Conf.*, pp. 131–142.

EDEN, M. (1961), On the formalization of handwriting. *Am. Math. Soc. Appl. Math. Symp.*, **12**, 83–88.

EDEN, M. (1962), Handwriting and pattern recognition. *IRE. Trans. Inform. Theory* **IT-8**, 160–166.

GRIMSDALE, R. L. *et al.* (1959), A system for the automatic recognition of patterns. *Proc. I. E. E.* **106**(B), 210.

MCCORMICK, B. H. (1963), The Illinois Pattern Recognition Computer (Illiac III). Digital Computer Laboratory Report No. 148, Univ. of Illinois.

MINSKY, M. (1961), Steps toward artificial intelligence. *Proc. I. R. E.* **49**, 8–30.

NARASIMHAN, R. (1962), A linguistic approach to pattern recognition. Digital Computer Laboratory Report No. 121, Univ. of Illinois.

NARASIMHAN, R. AND MAYOH, B. H. (1963), The structure of a program for scanning bubble chamber negatives. Digital Computer Laboratory File No. 507, Univ. of Illinois.

STEIN, J. H. (1963), Users' Manual for PAX, An IBM 7090 Program to Simulate the Pattern Articulation Unit of Illiac III. Digital Computer Laboratory Report No. 147, Univ. of Illinois.